Recommendations from CMR Cloud Prototype

Goal

The goal of the initial CMR Cloud Prototyping was to make three architectural determinations:

1. Determine feasibility of using Oracle RDS with the metadata either in S3 or stored as BLOB data within Oracle
2. Determine best storage to use for Elasticsearch - either local SSD storage or EBS general purpose SSD storage.
3. Determine the best storage to use for RabbitMQ - either local SSD storage or EBS general purpose SSD storage.

Traceability

CMR Cloud Phase 1 Task

Prototyping Summary

In this prototyping effort we conducted several tests to evaluate different solutions for Oracle, Elasticsearch, and RabbitMQ to assist in making the determinations outlined in the Goal section. The test methodologies and results are described in this paper and conclusions are presented. These are summarized as recommendations for each of the three architectural determinations.

In addition to obtaining information related to the primary goal we gained valuable experience utilizing the AWS environment. We learned how best to run our applications, ways to automate setup in AWS, as well as approaches and techniques for taking advantage of AWS that would not be possible on our own infrastructure. This experience is summarized in a separate lessons learned document.

Decision criteria for choosing the best option

Each alternative architecture was evaluated against the following weighted criteria:

Criterion	Description
Performance	Amount of time an operation takes to execute.
Reliability	Evaluate how likely the solution is to be available, returning the correct results, and how quickly can we recover from a failure.
Schedule Impact	Evaluate the solution against the risk of causing a delay to the schedule or requiring a significant time to implement.
Cost	The total cost of the solution.

Each of the alternatives was scored between 1 and 5 for each of the evaluation criteria with 5 being the best score. A weighted score was then computed by multiplying the weights for each criteria by the criteria score and then summing these values together. The alternative with the highest weighted score was chosen as the recommended solution.

Oracle RDS using S3 or BLOB for metadata

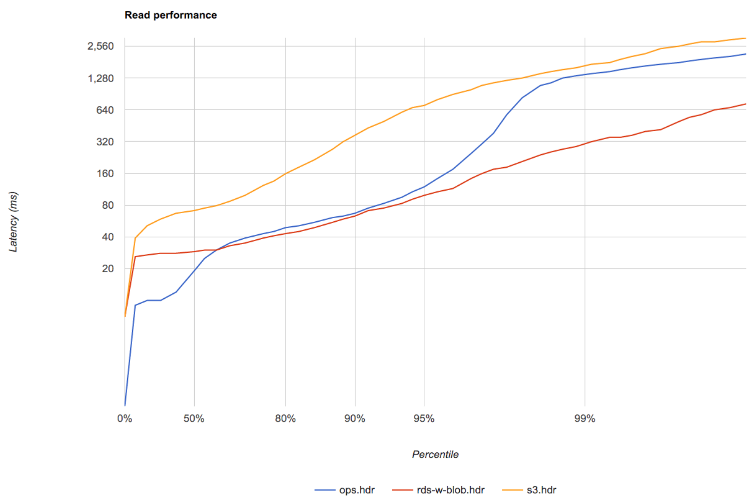
Performance Test Setup

We performed several tests writing to and reading from S3 with between 2,000 and 40,000,000 granules to understand how to achieve the lowest latency and best overall throughput using S3. We then loaded our full operational database into Oracle RDS and fully populated S3 with the metadata. In order to improve the performance of S3 we worked with our AWS representative to have them manually partition our bucket. After that we were able to write to S3 at a rate of 10,000 records per second.

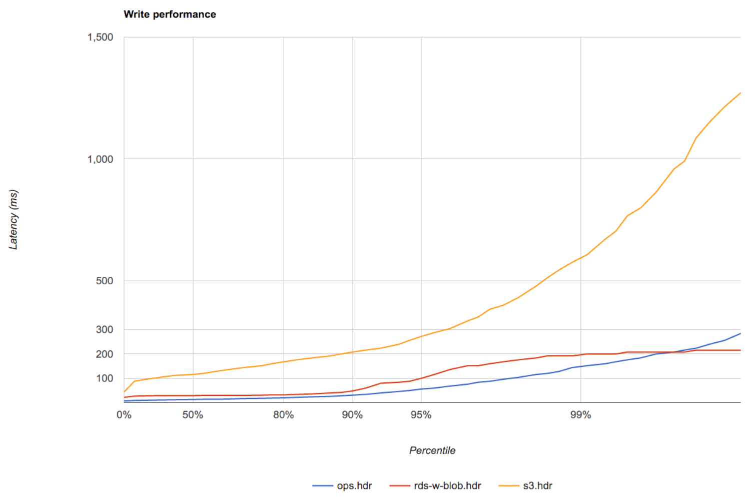
We set up our metadata-db application code to allow us to switch between using S3 or the Oracle BLOB column for the metadata. We then setup test drivers to write metadata and read metadata in a way comparable to the normal operations load with a total of 600K writes and 50K retrievals over 24 hours and compared the performance of the two solutions to operations. The results are summarized in the following charts.

Refer to the tickets in the epic ⚠ CMR-2589 - JIRA project doesn't exist or you don't have permission to view it. for detailed descriptions of all the tests performed and analysis of the results. The 2 figures below show the comparative read and write performance of the normal operational system (ops.hdr), Oracle RDS (rds-w-blob.hdr) with blobs, and Oracle RDS with S3 (s3.hdr).

Read Performance



Write Performance



Conclusion

Based on our testing we determined that Oracle RDS using BLOB storage for the metadata was the best option.

Criterion	Weight	S3	BLOB
Performance	5	3	5
Reliability	5	5	5
Schedule Impact	4	3	5
Cost	3	5	4
Weighted Score		**67**	**82**

While Oracle RDS using S3 proved to be scalable, the latency proved to be worse than our current operational performance and Oracle RDS using BLOB storage. It also scored lower on schedule risk because it requires additional code to be written for the bootstrap and metadata-db applications. Costs were slightly higher for Oracle RDS using BLOB storage for the metadata, however the performance benefit made it clear that it was the best solution.

Elasticsearch

Performance Test Setup

In order to test Elasticsearch we indexed our full operational holdings into Elasticsearch clusters in AWS. We setup two clusters each with 11 nodes with one cluster using EBS General Purpose SSD storage and EBS optimized instances and the other using local instance SSD storage. In our initial ROM we talked about having an additional replica resulting in 50% more data and a total 17 nodes as a risk mitigation to losing two nodes from the cluster without losing data, but for the prototyping we wanted to directly compare the performance of two clusters of the same size so we used the exact same configuration for each.

In order to drive the Elasticsearch testing we setup several CMR applications to run in AWS - cubby, index-set, indexer, metadata-db, and search and deployed them to 5 nodes like we have in operations. We then parsed our operational search logs to run the exact same set of queries over a 24 hour period. Finally we executed those tests against both Elasticsearch clusters. The full performance charts comparing ops performance, EBS, and ephemeral instance storage can be found here: ⚠ CMR-2645 - JIRA project doesn't exist or you don't have permission to view it.

At the end of those tests we found little difference in performance between the two types of storage. The slowest 0.2% of queries were faster with local instance storage, but otherwise there was little difference.

We executed a few other tests to determine if during certain scenarios the type of storage would make a difference. The only scenario where we observed a difference in performance was during a full S3 snapshot of the elasticsearch data while continuing to serve traffic. The snapshot took just under 45 minutes, during which time queries using the local storage were faster. A full snapshot will be rare in operations, however, since after the initial snapshot only deltas are recorded. Thereafter the snapshot process will complete much faster and we expect there to be little operational difference between the two storage solutions.

Refer to the tickets in the epic ⚠ CMR-2601 - JIRA project doesn't exist or you don't have permission to view it. for a detailed analysis of all of the tests performed.

Conclusion

Based on our testing we determined that Elasticsearch using EBS General Purpose SSD was the best option.

Criterion	Weight	EBS General Purpose SSD	Local SSD
Performance	5	5	5
Reliability	5	5	3
Schedule Impact	4	5	5
Cost	3	4	5
Weighted Score		**82**	**75**

We found that performance of Elasticsearch using both EBS General Purpose SSD and Local SSD storage was better than our current operational performance. While Local SSD storage had performance benefits in a couple of cases, the differences were so small that they both received the same score for performance. Local SSD storage has a small cost advantage over EBS General Purpose SSD storage.

The main discriminator between the two types of storage is that the storage backed by EBS general purpose SSD is persisted if the instance is stopped, while the data on a local storage instance is lost. The time to recovery of all primary shards if a cluster goes down with EBS General Purpose SSD storage is one minute and all replicas are available in 20 minutes. With local storage it takes just over an hour and a half to restore all primary shards from a snapshot, and three hours for full replica availability.

Recovery is necessary if two nodes in a cluster are lost. We talked about a mitigation of adding an additional replica so that recovery is needed instead when three nodes in a cluster are lost, however that requires doubling the number of nodes and causes costs of local storage to be more than EBS General Purpose SSD storage. In the end the better reliability and recovery scenarios is the driver for recommending EBS General Purpose SSD storage for Elasticsearch.

Message Queue

Introduction

Initially, we planned on testing RabbitMQ with both EBS General Purpose SSD storage and Local SSD storage, but based on the Elasticsearch tests it was clear that EBS General Purpose SSD storage provided better performance than our current operation systems and furthermore, the I/O requirements of our operational RabbitMQ servers are rather low. We therefore concluded that EBS General purpose SSD storage would be adequate for RabbitMQ.

As we started working with AWS more, we found that in general the Amazon provided services were excellent. They provide high availability, excellent stability, performance, scalability, and cost (particularly in comparison to running separate EC2 instances that provide equivalent functionality). In addition, services free up operations personnel from having to spin up, monitor, patch, and maintain our own EC2 instances, providing labor and cost savings. We read through the documentation on Amazon SQS (Simple Queue Service), discovered Amazon SNS (Amazon's pub/sub service), and realized that together they covered all of the functionality we use today with RabbitMQ. Therefore, we executed tests to determine if SNS/SQS could replace RabbitMQ in the CMR.

Performance Test Setup

We first added a new implementation of our Queue protocol to use SNS and SQS and setup our metadata-db and indexer applications to use the new implementation. We performed basic functionality tests to ensure all of our requirements were covered and then moved on to performance testing. Also note that the prototype code is really similar to the final production code we would need to use Amazon SNS and SQS, so there is little schedule risk related to implementation.

For performance testing we targeted processing messages at a rate of 10,000,000 requests per day which is the peak load we have simulated using RabbitMQ in our workload environment. For reference our normal operations ingest workload is around 600,000 requests per day. We used a test driver to submit 100 concurrent requests to a load balancer which submitted the requests to 5 different metadata-db applications. Metadata-db published the requests to SNS which populated the SQS queue. We started 5 indexer applications each with 5 threads pulling messages from the queue. We were able to sustain a rate of processing 135 requests per second for an hour (11,664,000 requests per day) with the same number of application servers we use today in operations. Messages were processed by the indexer as fast as they were coming in. Throughout the test we saw no more than 2 messages in flight on the queue. The 50th and 95th percentiles for individual publication requests were 48 and 119 ms respectively. Note that these times should be considered worst case as they were measured during the load test at a rate of 135 requests per second. For RabbitMQ in operations, publication request times at the 50th and 95th percentiles are 11 and 28 ms respectively. The total ingest request times in operations at the 50th and 95th percentiles are 127ms and 340ms respectively. Using SNS may increase ingest latency in the worst case by 29%.

Future Needs

We did not consider scalability as a scoring factor because all of the solutions can easily handle our current message queue workload which is driven by granule ingest. In the future scaling could be important as we introduce new concept types, supporting providers when they need to re-ingest all of their holdings quickly, and other use cases where CMR does not currently use message queues, but would in the future. RabbitMQ allows for a primary RabbitMQ instance and a backup instance that serves as a hot failover, but does not support a way to scale horizontally. On the other hand Amazon SNS and SQS can easily scale to handle orders of magnitude more messages than we use today.

Conclusion

Based on our testing we determined that replacing RabbitMQ with Amazon Simple Notification Service (SNS) and Amazon Simple Queue Service (SQS) was the best option.

Amazon SNS and SQS slightly outscored using RabbitMQ with EBS General Purpose SSD storage. After considering the future benefits of scalability and maintenance savings ever year it becomes even more apparent that using Amazon SNS and SQS is the best solution.

Criterion	Weight	RabbitMQ - EBS General Purpose SSD	RabbitMQ - Local SSD	Amazon SNS and SQS
Performance	3	5	5	4
Reliability	5	4	2	5
Schedule Impact	4	5	5	4
Cost	3	3	3	5
Weighted Score		**64**	**54**	**68**

Final Thoughts

While these are our recommendations based on the results of the prototyping, we can always revisit any of these decisions in the future as circumstances change. As part of the prototyping we discovered that is easy to test and deploy new configurations and solutions within AWS. For the message queues, we will still have a queue protocol implementation that supports RabbitMQ and can switch to using it with configuration. If the EBS storage costs for Elasticsearch became prohibitive we could build a new Elasticsearch cluster at any time using local storage and switch to it. Switching from Oracle BLOBs storage to using S3 can be done without a large effort. None of these solutions put us in a position that would require a large amount of effort to reverse.

Error rendering macro 'pageapproval' : null